INFORMATION AND COMPUTATION 112, 131–165 (1994)

Bounded Quantification Is Undecidable[1]

BENJAMIN C. PIERCE

LFCS, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, United Kingdom

$F_{\leqslant}$ is a typed λ-calculus with subtyping and bounded second-order polymorphism. First introduced by Cardelli and Wegner, it has been widely studied as a core calculus for type systems with subtyping. We use a reduction from the halting problem for two-counter Turing machines to show that the subtyping and typing relations of $F_{\leqslant}$ are undecidable.

1. INTRODUCTION

The notion of *bounded quantification* was introduced by Cardelli and Wegner (1985) in the language Fun. Based on informal ideas by Cardelli and formalized using techniques developed by Mitchell (1988), Fun integrated Girard–Reynolds polymorphism (Girard, 1972; Reynolds, 1974) and Cardelli's first-order calculus of subtyping (1988a).

Fun and its relatives have been studied extensively by programming language theorists and designers. Curien and Ghelli (1992) and Ghelli (1990) address a number of syntactic properties of the pure system. Semantic aspects of related systems have been studied by Bruce and Longo (1990), Martini (1988), Breazu-Tannen *et al.* (1991), Cardone (1989), Cardelli and Longo (1991), Cardelli *et al.* (1991), Curien and Ghelli (1991, 1992), and Bruce and Mitchell (1992). Pure $F_{\leqslant}$ has been extended to include record types and richer forms of polymorphism by Cardelli (1988b), Cardelli and Mitchell (1991), Bruce (1991, 1992, 1993), Cardelli (1992), Canning *et al.* (1989b), Ghelli (1991), Pierce and Turner (1994), and Hofmann and Pierce (1992). An extension with intersection types has been studied by the present author (Pierce, 1991, 1993). The proof theory of $F_{\leqslant}$ with a rule of extensionality has been studied by Curien and Ghelli (1991). The extension of $F_{\leqslant}$ with bounded existential types, first proposed by Cardelli and Wegner (1985), has been analyzed in more detail by Ghelli and Pierce (1992). An extension with recursive types has been studied by

[1] An identical version of this paper appears in *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*, Carl A. Gunter and John C. Mitchell, editors, MIT Press, 1994, by permission of Academic Press, Inc.

0890-5401/94 $6.00

Ghelli (1993). Bounded quantification plays a key role in Cardelli's programming language Quest (Cardelli, 1991b; Cardelli and Longo, 1991), in the Abel language developed at HP Labs (Canning *et al.*, 1989a, 1989b; Cook *et al.*, 1990), in the Rapide prototyping language under development at Stanford (Mitchell *et al.*, 1991), and in Bruce's object-oriented language TOOPLE (Bruce, 1993). Cardelli (1991a) has proposed a partial type inference method for programming languages based on $F_{\leqslant}$.

The original Fun was simplified by Bruce and Longo (1990) for their investigation of its semantics, and again by Curien and Ghelli (1992), who gave a proof of the coherence of typechecking. Curien and Ghelli's formulation, called *minimal Bounded Fun* or $F_{\leqslant}$ ("F sub"), is the one considered here.

The well-typed terms of $F_{\leqslant}$ are defined by means of a collection of rules for inferring statements of the form $\Gamma \vdash e \in \tau$ ("e has type τ under assumptions Γ"). Variables, abstractions, and applications have the usual typing rules:

$$\Gamma \vdash x \in \Gamma(x)$$

$$\frac{\Gamma, x{:}\sigma \vdash e \in \tau}{\Gamma \vdash \lambda x{:}\sigma.\, e \in \sigma \to \tau}$$

$$\frac{\Gamma \vdash e_1 \in \sigma \to \tau \quad \Gamma \vdash e_2 \in \sigma}{\Gamma \vdash e_1 e_2 \in \tau}$$

Type abstractions are treated as in other second-order λ-calculi, except that they also give a bound, with respect to the *subtype* relation, for the variable they introduce; they are checked by moving this assumption into the context and checking the body of the abstraction under the enriched set of assumptions:

$$\frac{\Gamma, \alpha \leqslant \theta \vdash e \in \tau}{\Gamma \vdash \Lambda\alpha \leqslant \theta.\, e \in \forall\alpha \leqslant \theta.\, \tau}$$

Type applications check that the type being passed as a parameter is indeed a subtype of the bound of the corresponding quantifier:

$$\frac{\Gamma \vdash e \in \forall\alpha \leqslant \theta.\, \tau \quad \Gamma \vdash \sigma \leqslant \theta}{\Gamma \vdash e[\sigma] \in \{\sigma/\alpha\}\tau}$$

Finally, like other λ-calculi with subtyping, $F_{\leqslant}$ includes a rule of *subsumption* allowing the type of a term to be promoted to any supertype:

$$\frac{\Gamma \vdash e \in \sigma \quad \Gamma \vdash \sigma \leqslant \tau}{\Gamma \vdash e \in \tau}$$

The rules for type application and subsumption rely on a separately axiomatized subtype relation $\Gamma \vdash \sigma \leqslant \tau$ ("σ is a subtype of τ under assumptions Γ"). This relation, which forms the main object of study in the present paper, is presented as follows. Subtyping is both reflexive and transitive:

$$\Gamma \vdash \tau \leqslant \tau$$

$$\frac{\Gamma \vdash \tau_1 \leqslant \tau_2 \quad \Gamma \vdash \tau_2 \leqslant \tau_3}{\Gamma \vdash \tau_1 \leqslant \tau_3}$$

Every type is a subtype of a maximal type called Top:

$$\Gamma \vdash \sigma \leqslant \text{Top}$$

(One of the main uses of Top—indeed, the reason it was originally introduced by Cardelli and Wegner—is to recover ordinary unbounded quantification as a special case of bounded quantification: $\forall\alpha.\,\tau$ becomes $\forall\alpha\leqslant\text{Top}.\,\tau$.) Type variables are subtypes of their bounds:

$$\Gamma \vdash \alpha \leqslant \Gamma(\alpha)$$

The subtype relation between arrow types is contravariant in their left-hand sides and covariant in their right-hand sides:

$$\frac{\Gamma \vdash \tau_1 \leqslant \sigma_1 \quad \Gamma \vdash \sigma_2 \leqslant \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leqslant \tau_1 \to \tau_2}$$

Similarly, subtyping of quantified types is contravariant in their bounds and covariant in their bodies:

$$\frac{\Gamma \vdash \tau_1 \leqslant \sigma_1 \quad \Gamma, \alpha \leqslant \tau_1 \vdash \sigma_2 \leqslant \tau_2}{\Gamma \vdash \forall\alpha\leqslant\sigma_1.\,\sigma_2 \leqslant \forall\alpha\leqslant\tau_1.\,\tau_2}$$

The last rule deserves a closer look, since it is the principal source of the difficulties we will be discussing for the rest of the paper. Intuitively, it reads as follows:

> A type $\tau \equiv \forall\alpha\leqslant\tau_1.\,\tau_2$ describes a collection of polymorphic values (functions from types to values), each mapping subtypes of τ_1 to instances of τ_2. If τ_1 is a subtype of σ_1, then the domain of τ is smaller than that of $\sigma \equiv \forall\alpha\leqslant\sigma_1.\,\sigma_2$, so τ is a weaker constraint and describes a larger collection of polymorphic values. Moreover, if, for each type θ that is an acceptable argument to the functions in both collections (i.e., one that satisfies the more stringent requirement $\theta \leqslant \tau_1$), the θ instance of σ_2 is a subtype of the θ instance of

τ_2, then τ is a "pointwise weaker" constraint and again describes a larger collection of polymorphic values.

Though semantically appealing, this rule creates serious problems for reasoning about the subtype relation. In a quantified type $\forall\alpha\leqslant\sigma_1.\,\sigma_2$, free occurrences of α in σ_2 are naturally thought of as being bounded by their lexically apparent bound σ_1. But this connection is destroyed by the second premise of the quantifier subtyping rule: when $\forall\alpha\leqslant\sigma_1.\,\sigma_2$ is compared to $\forall\alpha\leqslant\tau_1.\,\tau_2$, instances of α in *both* σ_2 and τ_2 are bounded by τ_1 in the premise $\Gamma, \alpha\leqslant\tau_1 \vdash \sigma_2 \leqslant \tau_2$. As we shall see, this "re-bounding" behavior makes it impossible to give a decision procedure for the subtype relation.

Cardelli and Wegner's original definition of Fun (1985) used a weaker quantifier subtyping rule, where $\forall\alpha\leqslant\sigma_1.\,\sigma_2$ is a subtype of $\forall\alpha\leqslant\tau_1.\,\tau_2$ only if σ_1 and τ_1 are identical; this variant of the system can easily be shown to be decidable. Later authors, including Cardelli, have worked with the more powerful formulation studied here.

Curien and Ghelli used a proof-normalization argument to show that $F_{\leqslant}$ typechecking is *coherent*—that is, that all derivations of a statement $\Gamma \vdash e \in \tau$ have the same interpretation in any semantics satisfying certain conditions. One corollary of their proof is the soundness and completeness of a natural syntax-directed procedure for computing minimal typings of $F_{\leqslant}$ terms, and a subroutine for checking the subtype relation; the same procedure had been developed by the group at Penn and by Cardelli for use in his Quest typechecker (Gunter, personal communication, 1990). The termination of Curien and Ghelli's typechecking procedure is equivalent to the termination of the subtyping algorithm. Ghelli (1990) proposed a proof of termination; but this proof was discovered—by Curien and Reynolds, independently—to contain a subtle mistake. In fact, Ghelli discovered that there are inputs for which the subtyping algorithm does *not* terminate (1992). Worse yet, these cases did not seem amenable to any simple form of cycle detection: when presented with one of them, the algorithm would generate an infinite sequence of different recursive calls with larger and larger contexts. This discovery reopened the question of the decidability of $F_{\leqslant}$.

The undecidability result presented here began as an attempt to formulate a more refined algorithm capable of detecting the kinds of divergence that could be induced in the simpler one. A series of partial results about decidable subsystems eventually led to the discovery of a class of input problems in which increasing the size the input by a constant factor would increase the search depth of a *succeeding* execution of the algorithm by an exponential factor. In addition to dispelling earlier intuitions about why the problem ought to be decidable, the technique used to construct this example suggested a trick for encoding natural

numbers, from which it was a short step to an encoding of two-counter Turing machines.

After formally defining the $F_{\leqslant}$ subtype relation (Section 2), reviewing Curien and Ghelli's subtyping algorithm (Section 3), and presenting an example where the algorithm fails to terminate (Section 4), we identify a fragment of $F_{\leqslant}$ that forms a convenient target for the reductions to follow (Sections 5 and 6). The main result is then presented in two steps:

1. We first define an intermediate abstraction, called *rowing machines* (Section 7). These machines bridge the gap between $F_{\leqslant}$ subtyping problems and two-counter machines by retaining the notions of bound variables and substitution from $F_{\leqslant}$ while introducing a computational abstraction with a finite collection of registers and an evaluation regime based on state transformation.

An encoding of rowing machines as $F_{\leqslant}$ subtyping statements is given and proven correct, in the sense that a rowing machine R halts iff its translation $\mathscr{F}(R)$ is a derivable statement in $F_{\leqslant}$ (Section 8).

2. We then review the standard definition of *two-counter machines* (Section 9) and show how a two-counter machine T may be encoded as a rowing machine $\mathscr{R}(T)$ such that T halts iff $\mathscr{R}(T)$ does (Section 10).

Section 11 shows that the undecidability of subtyping implies the undecidability of typechecking. Section 12 discusses the pragmatic import of these results.

2. The Subtype Relation

We begin the detailed development of the undecidability of $F_{\leqslant}$ by establishing some notational conventions and defining the subtype relation formally.

2.1. *Notation.* We write $X \equiv Y$, where X and Y are types, contexts, statements, etc., to indicate that "X has the form Y." If Y contains free metavariables, then $X \equiv Y$ denotes pattern matching; for example

$$\text{"If } \tau \equiv \forall \alpha \leqslant \tau_1 . \tau_2, \text{ then..."}$$

means

"If τ has the form $\forall \alpha \leqslant \tau_1 . \tau_2$ for some α, τ_1, and τ_2, then...".

2.2. Definition. The *types* of $F_{\leqslant}$ are defined by the following abstract grammar:

$$\begin{array}{lll} \tau ::= & \alpha & \text{type variables} \\ & \mid \tau_1 \to \tau_2 & \text{function types} \\ & \mid \forall \alpha \leqslant \tau_1 . \tau_2 & \text{bounded quantifiers} \\ & \mid \text{Top} & \text{top type.} \end{array}$$

The scope of the bound variable α in $\forall \alpha \leqslant \tau_1 . \tau_2$ is the body τ_2.

2.3. DEFINITION. *Typing contexts* in $F_{\leqslant}$ are finite sequences of type variables and associated bounds

$$\Gamma ::= \{\} \mid \Gamma, \alpha \leqslant \tau$$

with all variables distinct. (If we were dealing formally with the $F_{\leqslant}$ typing relation, we would also need bindings of the form $x{:}\tau$.)

The comma operator is used to denote both extension $(\Gamma, \alpha \leqslant \tau)$ and concatenation (Γ_1, Γ_2) of contexts. The set of variables bound by a context Γ is written $\text{dom}(\Gamma)$. When $\Gamma \equiv \Gamma_1, \alpha \leqslant \tau, \Gamma_2$, we call τ the *bound* of α in Γ and write $\tau = \Gamma(\alpha)$.

2.4. DEFINITION. A *subtyping statement* is a phrase of the form

$$\Gamma \vdash \sigma \leqslant \tau.$$

The portion of a statement to the right of the turnstile is called the *body*.

2.5. DEFINITION. The set of free type variables in a type τ is written $\text{FTV}(\tau)$.

2.6. DEFINITION. A type τ is *closed* with respect to a context Γ if $\text{FTV}(\tau) \subseteq \text{dom}(\Gamma)$. A context Γ is closed if

1. $\Gamma \equiv \{\}$, or
2. $\Gamma \equiv \Gamma_1, \alpha \leqslant \tau$, with Γ_1 closed and τ closed with respect to Γ_1.

A statement $\Gamma \vdash \sigma \leqslant \tau$ is closed if Γ is closed and σ and τ are closed with respect to Γ.

2.7. *Convention.* In the following, we assume that all statements under discussion are closed. In particular, we allow only closed statements in instances of inference rules.

2.8. *Convention.* Types, contexts, and statements that differ only in the names of bound variables are considered to be identical. (In a statement of the form $\Gamma_1, \alpha \leqslant \theta, \Gamma_2 \vdash \sigma \leqslant \tau$, the variable α is bound in Γ_2, σ, and τ.)

It is formally clearer to think of variables not as names but as pointers into the surrounding context, as suggested by de Bruijn (1972). This point of view is notationally too inconvenient to adopt explicitly in what follows, but it can be a significant aid in understanding the behavior of the rules here (VAR and ALL) that manipulate variables.

2.9. *Convention.* The metavariables σ, τ, θ, and ϕ range over types; α, β, and γ range over type variables; Γ ranges over contexts; J ranges over (closed) statements.

2.10. DEFINITION. $F_{\leqslant}$ is the least three-place relation closed under the following rules:

$$\Gamma \vdash \tau \leqslant \tau \qquad \text{(REFL)}$$

$$\frac{\Gamma \vdash \tau_1 \leqslant \tau_2 \quad \Gamma \vdash \tau_2 \leqslant \tau_3}{\Gamma \vdash \tau_1 \leqslant \tau_3} \qquad \text{(TRANS)}$$

$$\Gamma \vdash \sigma \leqslant \text{Top} \qquad \text{(TOP)}$$

$$\Gamma \vdash \alpha \leqslant \Gamma(\alpha) \qquad \text{(VAR)}$$

$$\frac{\Gamma \vdash \tau_1 \leqslant \sigma_1 \quad \Gamma \vdash \sigma_2 \leqslant \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leqslant \tau_1 \to \tau_2} \qquad \text{(ARROW)}$$

$$\frac{\Gamma \vdash \tau_1 \leqslant \sigma_1 \quad \Gamma, \alpha \leqslant \tau_1 \vdash \sigma_2 \leqslant \tau_2}{\Gamma \vdash \forall \alpha \leqslant \sigma_1 . \sigma_2 \leqslant \forall \alpha \leqslant \tau_1 . \tau_2} \qquad \text{(ALL)}$$

2.11. DEFINITION. The capture-avoiding substitution of σ for α in τ is written $\{\sigma/\alpha\}\tau$. Substitution is extended pointwise to contexts: $\{\sigma/\alpha\}\Gamma$.

2.12. DEFINITION. In the examples below, it will be convenient to rely on a few abbreviations:

$$\forall \alpha . \tau \quad \stackrel{\text{def}}{=} \quad \forall \alpha \leqslant \text{Top} . \tau$$

$$\forall \alpha_1 \leqslant \phi_1 .. \alpha_n \leqslant \phi_n . \tau \quad \stackrel{\text{def}}{=} \quad \forall \alpha_1 \leqslant \phi_1 \ldots \forall \alpha_n \leqslant \phi_n . \tau$$

$$\neg \tau \quad \stackrel{\text{def}}{=} \quad \forall \alpha \leqslant \tau . \alpha$$

The salient property of the last of these is that it allows the right- and left-hand sides of subtyping statements to be swapped:

2.13. *Fact.* $\Gamma \vdash \neg \sigma \leqslant \neg \tau$ is derivable iff $\Gamma \vdash \tau \leqslant \sigma$ is.

3. A Subtyping Algorithm

The rules defining $F_{\leqslant}$ do not constitute an algorithm for checking the subtype relation, since they are not syntax-directed. In particular, the Trans rule cannot effectively be applied backwards, since this would involve "guessing" an appropriate intermediate type τ_2. Curien and Ghelli (as well as Cardelli and others) use the following reformulation:

3.1. Definition. $F^N_{\leqslant}$ (N for normal form) is the least relation closed under the following rules:

$$\Gamma \vdash \sigma \leqslant \text{Top} \qquad \text{(NTop)}$$

$$\Gamma \vdash \alpha \leqslant \alpha \qquad \text{(NRefl)}$$

$$\frac{\Gamma \vdash \Gamma(\alpha) \leqslant \tau}{\Gamma \vdash \alpha \leqslant \tau} \qquad \text{(NVar)}$$

$$\frac{\Gamma \vdash \tau_1 \leqslant \sigma_1 \quad \Gamma \vdash \sigma_2 \leqslant \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 \leqslant \tau_1 \to \tau_2} \qquad \text{(NArrow)}$$

$$\frac{\Gamma \vdash \tau_1 \leqslant \sigma_1 \quad \Gamma, \alpha \leqslant \tau_1 \vdash \sigma_2 \leqslant \tau_2}{\Gamma \vdash \forall \alpha \leqslant \sigma_1.\, \sigma_2 \leqslant \forall \alpha \leqslant \tau_1.\, \tau_2} \qquad \text{(NAll)}$$

The reflexivity rule here is restricted to type variables. The general rule of transitivity is replaced by the new rule for variables, NVar, which captures the effect of an instance of the original variable rule followed by a single instance of transitivity:

$$\frac{\Gamma \vdash \alpha \leqslant \Gamma(\alpha) \quad \Gamma \vdash \Gamma(\alpha) \leqslant \tau}{\Gamma \vdash \alpha \leqslant \tau}$$

Curien and Ghelli's proof normalization argument shows that this is the only essential use of transitivity in the original system:

3.2. Lemma (Curien and Ghelli, 1992). *The relations $F_{\leqslant}$ and $F^N_{\leqslant}$ coincide: a statement $\Gamma \vdash \sigma \leqslant \tau$ is derivable in $F_{\leqslant}$ iff it is derivable in $F^N_{\leqslant}$.*

3.3. Definition. If we choose NTop and NRefl in preference to NVar whenever possible, the rules defining $F^N_{\leqslant}$ form an algorithm (i.e., a deterministic, recursively defined procedure, not necessarily always terminating) for checking the subtype relation:

check($\Gamma \vdash \sigma \leqslant \tau$) =

if $\tau \equiv$ Top

then true

else if $\sigma \equiv \sigma_1 \to \sigma_2$ and $\tau \equiv \tau_1 \to \tau_2$

then check($\Gamma \vdash \tau_1 \leqslant \sigma_1$)

and check($\Gamma \vdash \sigma_2 \leqslant \tau_2$)

else if $\sigma \equiv \forall \alpha \leqslant \sigma_1 . \sigma_2$ and $\tau \equiv \forall \alpha \leqslant \tau_1 . \tau_2$

then check($\Gamma \vdash \tau_1 \leqslant \sigma_1$)

and check($\Gamma, \alpha \leqslant \tau_1 \vdash \sigma_2 \leqslant \tau_2$)

else if $\sigma \equiv \alpha$ and $\tau \equiv \alpha$

then true

else if $\sigma \equiv \alpha$

then check($\Gamma \vdash \Gamma(\alpha) \leqslant \tau$)

else

false.

We write $F^N_{\leqslant}$ to refer either to the algorithm or to the inference system.

The algorithm may be thought of as incrementally building a normal form derivation of a statement J, starting from the root and recursively building subderivations for the premises. By Lemma 3.2, if there is any derivation whatsoever of a statement J, there is one in normal form; the algorithm is guaranteed to recapitulate this derivation and halt in finite time.

3.4. *Fact* (Curien and Ghelli, 1992). If $\Gamma \vdash \sigma \leqslant \tau$ is derivable in $F_{\leqslant}$, then the algorithm $F^N_{\leqslant}$ halts and returns *true* when given this statement as input. If $\Gamma \vdash \sigma \leqslant \tau$ is not derivable in $F_{\leqslant}$, the algorithm either returns *false* or diverges.

4. Nontermination of the Algorithm

Ghelli (1992) dispelled the widely held belief that $F^N_{\leqslant}$ terminates on all inputs by discovering the following example.

4.1. Example. Let

$$\theta \equiv \forall \alpha . \neg (\forall \beta \leqslant \alpha . \neg \beta).$$

(Recall that $\neg\tau = \forall\alpha{\leqslant}\tau.\ \alpha$.) Then executing $F_{\leqslant}^{N}$ on the input problem

$$\alpha_0{\leqslant}\theta \vdash \alpha_0 \leqslant (\forall\alpha_1{\leqslant}\alpha_0.\ \neg\alpha_1)$$

leads to the following infinite sequence of recursive calls:

1.	$\alpha_0{\leqslant}\theta$	$\vdash \alpha_0$	$\leqslant \forall\alpha_1{\leqslant}\alpha_0.\ \neg\alpha_1$
2.	$\alpha_0 \leqslant \theta$	$\vdash \forall\alpha_1.\ \neg(\forall\alpha_2{\leqslant}\alpha_1.\ \neg\alpha_2)$	$\leqslant \forall\alpha_1{\leqslant}\alpha_0.\ \neg\alpha_1$
3.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0$	$\vdash \neg(\forall\alpha_2{\leqslant}\alpha_1.\ \neg\alpha_2)$	$\leqslant \neg\alpha_1$
4.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0$	$\vdash \alpha_1$	$\leqslant \forall\alpha_2{\leqslant}\alpha_1.\ \neg\alpha_2$
5.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0$	$\vdash \alpha_0$	$\leqslant \forall\alpha_2{\leqslant}\alpha_1.\ \neg\alpha_2$
6.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0$	$\vdash \forall\alpha_2.\ \neg(\forall\alpha_3{\leqslant}\alpha_2.\ \neg\alpha_3)$	$\leqslant \forall\alpha_2{\leqslant}\alpha_1.\ \neg\alpha_2$
7.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0,\ \alpha_2{\leqslant}\alpha_1$	$\vdash \neg(\forall\alpha_3{\leqslant}\alpha_2.\ \neg\alpha_3)$	$\leqslant \neg\alpha_2$
8.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0,\ \alpha_2{\leqslant}\alpha_1$	$\vdash \alpha_2$	$\leqslant \forall\alpha_3{\leqslant}\alpha_2.\ \neg\alpha_3$
9.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0,\ \alpha_2{\leqslant}\alpha_1$	$\vdash \alpha_1$	$\leqslant \forall\alpha_3{\leqslant}\alpha_2.\ \neg\alpha_3$
10.	$\alpha_0{\leqslant}\theta,\ \alpha_1{\leqslant}\alpha_0,\ \alpha_2{\leqslant}\alpha_1$	$\vdash \alpha_0$	$\leqslant \forall\alpha_3{\leqslant}\alpha_2.\ \neg\alpha_3$

etc.

The α-conversion steps necessary to maintain the well-formedness of the context when new variables are added are performed tacitly here, choosing new names so as to clarify the pattern of regress. The crucial trick is the "re-bounding" that occurs, for instance, between steps 2 and 3, where the bound of α_1 on the left-hand side is changed from Top in line 2 to α_0 in line 3. Since the whole left-hand side in line 2 is itself the upper-bound of α_0, the re-bounding creates a cyclic pattern where longer and longer chains of variables in the context must be traversed on each loop.

The reader is cautioned not to look for *semantic* intuitions behind this example; in particular, $\neg\tau$ is a negation only in the sense that it allows the left- and right-hand sides of subtyping judgements to be swapped. From this point on, we will be dealing with purely combinatorial arguments concerning the proof theory of $F_{\leqslant}$.

5. A Deterministic Fragment of $F_{\leqslant}$

The pattern of recursion in Ghelli's example is an instance of a more general scheme—one so general, in fact, that it can be used to encode termination problems for two-counter Turing machines. We now turn to demonstrating this fact.

5.1. DEFINITION. The *positive* and *negative occurrences* in a statement $\Gamma \vdash \sigma \leqslant \tau$ are defined as follows:

- The type σ and the bounds in Γ are negative occurrences; τ is a positive occurrence.
- If $\tau_1 \to \tau_2$ is a positive (respectively, negative) occurrence, then τ_1 is a negative (positive) occurrence and τ_2 is a positive (negative) occurrence.
- If $\forall \alpha \leqslant \tau_1 . \tau_2$ is a positive (negative) occurrence, then τ_1 is a negative (positive) occurrence and τ_2 is a positive (negative) occurrence.

5.2. *Fact.* The rules defining $F_{\leqslant}^{N}$ preserve the signs of occurrences: wherever a metavariable τ appears in a premise of one of the rules, it has the same sign as the corresponding occurrence of τ in the conclusion.

In what follows, it will be convenient to work in a fragment of $F_{\leqslant}^{N}$ with somewhat simpler behavior:

- we drop the $\to$ type constructor and its subtyping rule;
- we introduce the negation operator explicitly into the syntax and include a rule for comparing negated expressions;
- we drop the left-hand premise from the rule for comparing quantifiers, requiring instead that when two quantified types are compared, the bound of the one on the left must be Top;
- we consider only statements where no variable occurs positively, allowing us to drop the NREFL rule; and
- we disallow Top in negative positions.

Since the $F_{\leqslant}^{N}$ rules preserve positive and negative occurrences, we may redefine the set of types so that positive and negative types form separate syntactic categories. At the same time, we simplify each category appropriately.

5.3. DEFINITION. The sets of *positive types* τ^+ and *negative types* τ^- are defined by the following abstract grammar:

$$\tau^+ ::= \text{Top} \mid \neg \tau^- \mid \forall \alpha \leqslant \tau^- . \tau^+$$

$$\tau^- ::= \alpha \mid \neg \tau^+ \mid \forall \alpha . \tau^-$$

A *negative context* Γ^- is one whose bounds are all negative types.

5.4. DEFINITION. $F_{\leqslant}^{P}$ (P for polarized) is the least relation closed under the following rules:

$$\Gamma^- \vdash \tau^- \leqslant \text{Top} \tag{PTOP}$$

$$\frac{\Gamma^- \vdash \Gamma^-(\alpha) \leqslant \tau^+}{\Gamma^- \vdash \alpha \leqslant \tau^+} \tag{PVAR}$$

$$\frac{\Gamma^-, \alpha \leqslant \phi^- \vdash \sigma^- \leqslant \tau^+}{\Gamma^- \vdash \forall \alpha. \sigma^- \leqslant \forall \alpha \leqslant \phi^-. \tau^+} \tag{PALL}$$

$$\frac{\Gamma^- \vdash \tau^- \leqslant \sigma^+}{\Gamma^- \vdash \neg \sigma^+ \leqslant \neg \tau^-} \tag{PNEG}$$

$F^{\mathrm{P}}_{\leqslant}$ lacks one property that we will need in what follows: $F_{\leqslant}$ is not a conservative extension of $F^{\mathrm{P}}_{\leqslant}$. For example, the nonderivable $F^{\mathrm{P}}_{\leqslant}$ statement

$$\vdash \neg \text{Top} \leqslant \forall \alpha. \alpha$$

corresponds, under the abbreviations for $\neg$ and $\forall \alpha. \alpha$, to the derivable $F_{\leqslant}$ statement

$$\vdash \forall \alpha \leqslant \text{Top}. \alpha \leqslant \forall \alpha \leqslant \text{Top}. \alpha.$$

To achieve conservativity, we restrict the form of $F^{\mathrm{P}}_{\leqslant}$ statements even further so that negated types can never be compared with quantified types.

5.5. DEFINITION. Let n be a fixed nonnegative number. The sets of *n-positive* and *n-negative types* are defined by the following abstract grammar:

$$\tau^+ ::= \text{Top} \mid \forall \alpha_0 \leqslant \tau_0^- .. \alpha_n \leqslant \tau_n^- . \neg \tau^-$$

$$\tau^- ::= \alpha \mid \forall \alpha_0 .. \alpha_n . \neg \tau^+$$

An n-positive type $\forall \alpha_0 \leqslant \tau_0^- .. \alpha_n \leqslant \tau_n^- . \neg \tau^-$ is closed only if no α_i appears free in any τ_j: the α_i's are bound simultaneously. An *n-negative context* is one whose bounds are all n-negative types.

5.6. DEFINITION. An $F^{\mathrm{D}}_{\leqslant}$ *statement* (D for deterministic) has the form $\Gamma^- \vdash \sigma^- \leqslant \tau^+$, where, for some n, Γ^- is an n-negative context, σ^- is an n-negative type, and τ^+ is an n-positive type.

5.7. *Convention.* To reduce notational clutter, we drop the superscripts $+$ and $-$ and usually leave n implicit.

5.8. DEFINITION. $F^{\mathrm{D}}_{\leqslant}$ is the least relation closed under the following rules:

$$\Gamma \vdash \tau \leqslant \text{Top} \qquad \text{(DTop)}$$

$$\frac{\Gamma \vdash \Gamma(\alpha) \leqslant \forall\alpha_0\leqslant\phi_0 .. \alpha_n\leqslant\phi_n.\ \neg\tau}{\Gamma \vdash \alpha \leqslant \forall\alpha_0\leqslant\phi_0 .. \alpha_n\leqslant\phi_n.\ \neg\tau} \qquad \text{(DVar)}$$

$$\frac{\Gamma, \alpha_0\leqslant\phi_0 .. \alpha_n\leqslant\phi_n \vdash \tau \leqslant \sigma}{\Gamma \vdash \forall\alpha_0 .. \alpha_n.\ \neg\sigma \leqslant \forall\alpha_0\leqslant\phi_0 .. \alpha_n\leqslant\phi_n.\ \neg\tau} \qquad \text{(DAllNeg)}$$

Using the earlier abbreviations for negation, multiple quantification, and unbounded quantification, we may read every $F^{\mathrm{D}}_{\leqslant}$ statement as an $F^{\mathrm{N}}_{\leqslant}$ statement. The two subtype relations then coincide for statements in their common domain:

5.9. Lemma. *$F^{\mathrm{N}}_{\leqslant}$ is a conservative extension of $F^{\mathrm{D}}_{\leqslant}$: if J is an $F^{\mathrm{D}}_{\leqslant}$ statement, then J is derivable in $F^{\mathrm{D}}_{\leqslant}$ iff it is derivable in $F^{\mathrm{N}}_{\leqslant}$.*

Proof. The implication $F^{\mathrm{D}}_{\leqslant} \Rightarrow F^{\mathrm{N}}_{\leqslant}$ is straightforward. The other direction, $F^{\mathrm{N}}_{\leqslant} \Rightarrow F^{\mathrm{D}}_{\leqslant}$, proceeds by induction on $F^{\mathrm{N}}_{\leqslant}$ derivations whose conclusions are de-abbreviated $F^{\mathrm{D}}_{\leqslant}$ statements. ∎

These simplifications justify a useful change of perspective. Since each rule in $F^{\mathrm{D}}_{\leqslant}$ has at most one premise, derivations in this fragment are linear. The syntax-directed construction of such a derivation may be thus be viewed as a deterministic state transformation process, where the subtyping statement being verified is the current state and the single premise that must be recursively verified, if any, is the next state. In other words, a subtyping statement is thought of as an instantaneous description of a sort of automaton.

From now on, we use terminology that makes the intuition of "subtyping as state transformation" more explicit. Analogous terminology and notation will apply to the other calculi introduced below.

5.10. Definition. The *one-step elaboration* function $\mathscr{E}$ for $F^{\mathrm{D}}_{\leqslant}$-statements is the partial mapping

$$\mathscr{E}(J) = \begin{cases} J' & \text{if } J \text{ is the conclusion of an instance of DVar} \\ & \text{or DAllNeg and } J' \text{ is the corresponding premise} \\ \text{undefined} & \text{if } J \text{ is an instance of DTop.} \end{cases}$$

(Note that, unlike $F^{\mathrm{N}}_{\leqslant}$, every $F^{\mathrm{D}}_{\leqslant}$ statement is an instance of the conclusion of one of the three rules; the algorithm may either return *true* or diverge, but it never returns *false*.)

5.11. Definition. J' is an *immediate subproblem* of J in $F^{\mathrm{D}}_{\leqslant}$, written $J \longrightarrow_{\mathrm{D}} J'$, if $J' = \mathscr{E}(J)$.

5.12. DEFINITION. J' is a *subproblem* of J in $F^{D}_{\leqslant}$, written $J \xrightarrow{*}_{D} J'$, if either $J \equiv J'$ or $J \longrightarrow_{D} J_1$ and $J_1 \xrightarrow{*}_{D} J'$.

5.13. DEFINITION. J' is a *proper subproblem* of J in $F^{D}_{\leqslant}$, written $J \xrightarrow{+}_{D} J'$, if $J \longrightarrow_{D} J_1$ and $J_1 \xrightarrow{*}_{D} J'$.

5.14. DEFINITION. The *elaboration* of a statement J is the sequence of subproblems encountered by a subtyping algorithm given J as input.

6. EAGER SUBSTITUTION

To make a smooth transition between the subtyping statements of $F_{\leqslant}$ and the rowing machine abstraction we will introduce in Section 7, we need one more variation in the definition of subtyping, where, instead of maintaining a context with the bounds of free variables, the quantifier rule immediately substitutes the bounds into the body of the statement.

6.1. DEFINITION. The simultaneous, capture-avoiding substitution of ϕ_0 through ϕ_n for α_0 through α_n in τ is written $\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\tau$.

6.2. DEFINITION. An $F^{F}_{\leqslant}$ *statement* (F for flattened) is an $F^{D}_{\leqslant}$ statement with an empty context.

6.3. DEFINITION. $F^{F}_{\leqslant}$ is the least relation closed under the following rules:

$$\vdash \tau \leqslant \mathrm{Top} \qquad \text{(FTop)}$$

$$\frac{\vdash \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\tau \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\sigma}{\vdash \forall\alpha_0 .. \alpha_n.\ \neg\sigma \leqslant \forall\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n.\ \neg\tau} \qquad \text{(FAllNeg)}$$

6.4. *Remark.* Of course, an analogous reformulation of full $F_{\leqslant}$ would not be correct. For example, in the nonderivable statement

$$\vdash (\forall\alpha \leqslant \mathrm{Top}.\ \mathrm{Top}) \leqslant (\forall\alpha \leqslant \mathrm{Top}.\ \alpha),$$

substituting Top for α in the bodies of the quantifiers yields the derivable statement

$$\vdash \mathrm{Top} \leqslant \mathrm{Top}.$$

But having restricted our attention to statements where variables appear only negatively, we are guaranteed that the only position where the elaboration of a statement can cause a variable to appear by itself in a sub-

problem is on the left-hand side, where it will immediately be replaced by its bound. We are therefore safe in making the substitution eagerly.

6.5. EXAMPLE. In this system, Ghelli's diverging example has a more periodic behavior:

$$\begin{array}{lll} 1. & \vdash \theta & \leqslant \forall\alpha_1\leqslant\theta.\ \neg\,\alpha_1 \\ \text{i.e.,} & \vdash \forall\alpha_1.\ \neg\,(\forall\alpha_2\leqslant\alpha_1.\ \neg\,\alpha_2) & \leqslant \forall\alpha_1\leqslant\theta.\ \neg\,\alpha_1 \\ 2. & \vdash \theta & \leqslant \forall\alpha_2\leqslant\theta.\ \neg\,\alpha_2 \end{array}$$

etc.

In the remainder of this section, we verify that $F^{\mathrm{D}}_{\leqslant}$ is a conservative extension of $F^{\mathrm{F}}_{\leqslant}$.

6.6. LEMMA. *Let $\phi_0..\phi_n$ be n-negative types and assume that the $F^{\mathrm{D}}_{\leqslant}$ statement $\alpha_0\leqslant\phi_0\,..\,\alpha_n\leqslant\phi_n,\ \Gamma\vdash\tau\leqslant\sigma$ is closed. Then if $\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\Gamma\vdash\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\tau\leqslant\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\sigma$ is derivable in $F^{\mathrm{D}}_{\leqslant}$, so is $\alpha_0\leqslant\phi_0\,..\,\alpha_n\leqslant\phi_n,\ \Gamma\vdash\tau\leqslant\sigma$.*

Proof. By induction on the size of the given derivation. Note that by the stipulation in 5.5 that no α_i appears in any ϕ_i, the ϕ_i must all be closed.

Case DTOP:

$$\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\sigma\equiv\mathrm{Top}.$$

Since variables can only occur negatively, σ cannot be a variable, so $\sigma\equiv\mathrm{Top}$ and the result is immediate.

Case DVAR:

$$\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\tau\equiv\beta.$$

We may assume that $\tau\not\equiv\alpha_i$ for any of the distinguished α_i's, since otherwise we would have $\phi_i\equiv\beta$ and the statement $\alpha_0\leqslant\phi_0\,..\,\alpha_n\leqslant\phi_n,\ \Gamma\vdash\tau\leqslant\sigma$ would not be closed. So τ must itself be β. By assumption, we have a subderivation

$$\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\Gamma\vdash(\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\Gamma)(\beta)\leqslant\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\sigma;$$

that is,

$$\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\Gamma\vdash\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}(\Gamma(\beta))\leqslant\{\phi_0/\alpha_0\,..\,\phi_n/\alpha_n\}\sigma.$$

By the induction hypothesis,

$$\alpha_0\leqslant\phi_0\,..\,\alpha_n\leqslant\phi_n,\ \Gamma\vdash\Gamma(\beta)\leqslant\sigma.$$

By DVAR,

$$\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma \vdash \beta \leqslant \sigma.$$

Case DALLNEG:

$$\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\tau \equiv \forall \beta_0 .. \beta_n. \neg \tau_2'$$
$$\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\sigma \equiv \forall \beta_0 \leqslant \psi_0' .. \beta_n \leqslant \psi_n'. \neg \sigma_2'.$$

Since σ cannot be a variable (else $\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma \vdash \tau \leqslant \sigma$ would not be an $F_{\leqslant}^{D}$ statement), we have

$$\sigma \equiv \forall \beta_0 \leqslant \psi_0 .. \beta_n \leqslant \psi_n. \neg \sigma_2$$
$$\psi_i' \equiv \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_i$$
$$\sigma_2' \equiv \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\sigma_2.$$

For τ, there are two cases to consider:

Subcase:

$$\tau \equiv \alpha_i.$$

Then

$$\phi_i \equiv \forall \beta_0 .. \beta_n. \neg \tau_2'.$$

By assumption, there is a subderivation

$$\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\Gamma, \beta_0 \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_0 .. \beta_n \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_n \vdash \sigma_2' \leqslant \tau_2';$$

i.e., (since we stipulated $\alpha_j \notin \mathrm{FTV}(\phi_i)$, so $\alpha_j \notin \mathrm{FTV}(\tau_i')$ for any j),

$$\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\Gamma, \beta_0 \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_0 .. \beta_n \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_n \vdash \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\sigma_2 \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\tau_2'.$$

By the induction hypothesis,

$$\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma, \beta_0 \leqslant \psi_0 .. \beta_n \leqslant \psi_n \vdash \sigma_2 \leqslant \tau_2'.$$

By DALLNEG,

$$\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma \vdash \forall \beta_0 .. \beta_n. \neg \tau_2' \leqslant \forall \beta_0 \leqslant \psi_0 .. \beta_n \leqslant \psi_n. \neg \sigma_2.$$

By DVAR,

$$\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma \vdash \alpha_i \leqslant \forall \beta_0 \leqslant \psi_0 .. \beta_n \leqslant \psi_n. \neg \sigma_2.$$

Subcase:

$$\tau \not\equiv \alpha_i.$$

Then

$$\tau \equiv \forall \beta_0 .. \beta_n. \neg \tau_2$$
$$\tau_2' \equiv \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\tau_2.$$

By assumption, we again have a subderivation

$$\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\Gamma, \beta_0 \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_0 .. \beta_n \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_n \vdash \sigma_2' \leqslant \tau_2';$$

that is,

$$\{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\Gamma, \beta_0 \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_0 .. \beta_n \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\psi_n \vdash \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\sigma_2 \leqslant \{\phi_0/\alpha_0 .. \phi_n/\alpha_n\}\tau_2.$$

By the induction hypothesis,

$$\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma, \beta_0 \leqslant \psi_0 .. \beta_n \leqslant \psi_n \vdash \sigma_2 \leqslant \tau_2.$$

By DALLNEG,

$$\alpha_0 \leqslant \phi_0 .. \alpha_n \leqslant \phi_n, \Gamma \vdash \forall \beta_0 .. \beta_n. \neg \tau_2 \leqslant \forall \beta_0 .. \beta_n. \neg \sigma_2. \quad \blacksquare$$

6.7. LEMMA. *If* $\vdash \sigma \leqslant \tau$ *is derivable in* $F^{F}_{\leqslant}$, *then it is derivable in* $F^{D}_{\leqslant}$.

Proof. By induction on the original derivation, using Lemma 6.6 for the FALLNEG case. ∎

6.8. LEMMA. *If* $\alpha \leqslant \phi$, $\Gamma \vdash \sigma \leqslant \tau$ *is derivable in* $F^{D}_{\leqslant}$, *then* $\{\phi/\alpha\}\Gamma \vdash \{\phi/\alpha\}\sigma \leqslant \{\phi/\alpha\}\tau$ *has an* $F^{D}_{\leqslant}$*-derivation of equal or lesser size.*

Proof. By induction on the given derivation. ∎

6.9. LEMMA. *If* $\vdash \sigma \leqslant \tau$ *is an* $F^{F}_{\leqslant}$*-statement and is derivable in* $F^{D}_{\leqslant}$, *then it is derivable in* $F^{F}_{\leqslant}$.

Proof. By induction on the size of the original derivation, using Lemma 6.8 for the DALLNEG case. ∎

6.10. LEMMA. $F^{D}_{\leqslant}$ *is a conservative extension of* $F^{F}_{\leqslant}$.

Proof. By Lemmas 6.7 and 6.9. ∎

7. Rowing Machines

The reduction from two-counter Turing machines to $F_{\leqslant}$ subtyping statements is easiest to understand in terms of an intermediate abstraction called a rowing machine, which makes more stylized use of bound variables.

A rowing machine is a tuple of *registers*

$$\langle \rho_1 .. \rho_n \rangle,$$

where the contents of each register is a *row*. By convention, the first register is the machine's *program counter* (PC). To move to the next state, the PC is used as a template to construct the new contents of each of the registers from the current contents of all of the registers (including the PC).

7.1. Definition. The set of *rows* of width n is defined by the following abstract grammar:

$$\begin{aligned} \rho ::= & \ \alpha \\ & \mid [\alpha_1..\alpha_n]\langle \rho_1..\rho_n \rangle \\ & \mid \text{HALT} \end{aligned}$$

The variables $\alpha_1..\alpha_n$ on the left of $[\alpha_1..\alpha_n]\langle \rho_1..\rho_n \rangle$ are binding occurrences whose scope is the rows ρ_1 through ρ_n. We regard rows that differ only in the names of bound variables as identical.

7.2. Definition. A *rowing machine* (of width n) is a tuple $\langle \rho_1..\rho_n \rangle$, where each ρ_i is a row of width n with no free variables.

7.3. Definition. The *one-step elaboration* function $\mathscr{E}$ for rowing machines of width n is the partial mapping

$$\mathscr{E}(\langle \rho_1..\rho_n \rangle) = \begin{cases} \langle \{\rho_1/\alpha_1 .. \rho_n/\alpha_n\}\rho_{11} .. \{\rho_1/\alpha_1 .. \rho_n/\alpha_n\}\rho_{1n} \\ \qquad \text{if} \quad \rho_1 = [\alpha_1..\alpha_n]\langle \rho_{11}..\rho_{1n} \rangle \\ \text{undefined} \\ \qquad \text{if} \quad \rho_1 = \text{HALT}. \end{cases}$$

(Since rowing machines consist only of closed rows, we need not define the evaluation function for the case where the PC is a variable. Also, since all the ρ_n are closed, the substitution is trivially capture-avoiding.)

7.4. *Notational Conventions.* 1. When the symbol "—" appears as the ith component of a compound row $[\alpha_1..\alpha_n]\langle \rho_1..\rho_n \rangle$, it stands for the variable α_i.

2. To avoid a proliferation of variable names in the examples and definitions below, we sometimes use numerical indices rather than names for variables: the "variable" $\#n$ refers to the nth bound variable of the row in which it appears; $\#\#n$ refers to the nth bound variable of the row enclosing the one in which it appears; and so on.

3. When these abbreviations are used, the bindings $[\alpha_1..\alpha_n]$ are omitted.

For example, the nested row

$$[\alpha_1..\alpha_3]\langle\alpha_1, [\beta_1..\beta_3]\langle\alpha_1, \beta_1, \beta_3\rangle, \alpha_1\rangle$$

is abbreviated as

$$\langle\text{—}, \langle \#\#1, \#1, \text{—}\rangle, \#1\rangle.$$

4. It is convenient to introduce names for closed rows and use these to build descriptions of other rows. For example, the compound row

$$\langle\langle\langle \#1, \#1, \#1\rangle, \#3, \#2\rangle, \langle\text{—}, \text{—}, \text{—}\rangle, \langle \#1, \#1, \#1\rangle\rangle$$

might be written as

$$\langle Z, Y, X\rangle,$$

where

$$X \equiv \langle \#1, \#1, \#1\rangle$$

$$Y \equiv \langle\text{—}, \text{—}, \text{—}\rangle$$

$$Z \equiv \langle X, \#3, \#2\rangle.$$

7.5. DEFINITION. A rowing machine R *halts* if there is a machine R' such that $R \xrightarrow{*}_R R'$ and the PC of R' is the instruction HALT.

7.6. EXAMPLE. The simplest rowing machine, $\langle$HALT$\rangle$, halts immediately. The next simplest, $\langle\langle$HALT$\rangle\rangle$, takes one step and then halts. Another simple one, $\langle\langle\text{—}\rangle\rangle$, leads to an infinite elaboration with every state identical to the first.

7.7. EXAMPLE. The machine

$$\langle\text{LOOP, A, B}\rangle,$$

where

$$\text{LOOP} \equiv \langle —, \#3, \#2 \rangle$$
$$\text{A} \equiv \text{an arbitrary row}$$
$$\text{B} \equiv \text{an arbitrary row,}$$

executes an infinite loop where the contents of the second and third register are exchanged at successive steps:

$$\begin{aligned} &\langle \text{LOOP, A, B} \rangle \\ \longrightarrow_R\ &\langle \text{LOOP, B, A} \rangle \\ \longrightarrow_R\ &\langle \text{LOOP, A, B} \rangle \\ \longrightarrow_R\ &\cdots \end{aligned}$$

7.8. EXAMPLE. The row

$$\text{BRI} \equiv \langle \#2, — \rangle$$

encodes an *indirect branch* to the contents of register 2 at the moment when BRI is executed—i.e., a branch "through register 2." The machine

$$\langle \text{BRI}, \langle \text{BRI}, \langle \text{BRI, HALT} \rangle \rangle \rangle$$

elaborates as follows:

$$\begin{aligned} &\langle \text{BRI}, \langle \text{BRI}, \langle \text{BRI, HALT} \rangle \rangle \rangle \\ \longrightarrow_R\ &\langle \langle \text{BRI}, \langle \text{BRI, HALT} \rangle \rangle, \langle \text{BRI}, \langle \text{BRI, HALT} \rangle \rangle \rangle \\ \longrightarrow_R\ &\langle \text{BRI}, \langle \text{BRI, HALT} \rangle \rangle \\ \longrightarrow_R\ &\langle \langle \text{BRI, HALT} \rangle, \langle \text{BRI, HALT} \rangle \rangle \\ \longrightarrow_R\ &\langle \text{BRI, HALT} \rangle \\ \longrightarrow_R\ &\langle \text{HALT, HALT} \rangle. \end{aligned}$$

8. ENCODING ROWING MACHINES AS SUBTYPING PROBLEMS

We now show how a rowing machine R can be encoded as a subtyping problem $\mathscr{F}(R)$ such that R halts iff $\mathscr{F}(R)$ is derivable in $F^{\text{F}}_{\leqslant}$. The idea of the translation is that a rowing machine $R = \langle \rho_1 .. \rho_n \rangle$ becomes a subtyping statement of the form

$$\vdash \ldots \leqslant (\ldots \mathscr{F}(\rho_1) \ldots)$$

(we use $\mathscr{F}$ to denote the translation of both rowing machines and rows), constructed so that

- if $\rho_1 = \text{HALT}$, then the elaboration of $\mathscr{F}(R)$ halts (by reaching a subproblem where Top appears on the right-hand side);
- if $\rho_1 = [\alpha_1..\alpha_n]\langle\rho_{11}..\rho_{1n}\rangle$, then the elaboration of $\mathscr{F}(R)$ reaches a subproblem that encodes the rowing machine $\mathscr{E}(\langle\rho_1..\rho_n\rangle) = \langle\{\rho_1/\alpha_1 .. \rho_n/\alpha_n\}\rho_{11} .. \{\rho_1/\alpha_1 .. \rho_n/\alpha_n\}\rho_{1n}\rangle$.

In more detail, if $R = \langle[\alpha_1..\alpha_n]\langle\rho_{11}..\rho_{1n}\rangle..\rho_n\rangle$, then $\mathscr{F}(R)$ is

$$\vdash\ \forall\gamma_1..\gamma_n.\ \neg(\forall\gamma_1'\leqslant\gamma_1 .. \gamma_n'\leqslant\gamma_n.\ \neg\ldots)$$

$$\leqslant\ \forall\gamma_1\leqslant\mathscr{F}(\rho_1) .. \gamma_n\leqslant\mathscr{F}(\rho_n).\ \neg(\forall\alpha_1..\alpha_n.\ \neg(\forall\alpha_1'\leqslant\mathscr{F}(\rho_{11}) .. \alpha_n'\leqslant\mathscr{F}(\rho_{1n}).\ \neg\mathscr{F}(\rho_{11}))).$$

The elaboration of this statement proceeds as follows:

1. The current contents of the registers $\rho_1..\rho_n$ are temporarily saved by matching the quantifiers on the right with the ones on the left; this has the effect of substituting the bounds $\mathscr{F}(\rho_1) .. \mathscr{F}(\rho_n)$ for free occurrences of the variables $\gamma_1..\gamma_n$ on the left-hand side.

The right- and left-hand sides are also swapped (by the $\neg$ constructor on both sides), so that what now appears on the left is a sequence of quantifiers binding the free variables $\alpha_1..\alpha_n$ of ρ_1:

$$\vdash\ \forall\alpha_1..\alpha_n.\ \neg(\forall\alpha_1'\leqslant\mathscr{F}(\rho_{11}) .. \alpha_n'\leqslant\mathscr{F}(\rho_{1n}).\ \neg\mathscr{F}(\rho_{11}))$$

$$\leqslant\ \forall\gamma_1'\leqslant\mathscr{F}(\rho_1) .. \gamma_n'\leqslant\mathscr{F}(\rho_n).\ \neg\ldots$$

2. The saved contents of the original registers now appear on the right-hand side. When these are matched with the quantifiers on the left, the result is that the old values of the registers are substituted for the variables $\alpha_1..\alpha_n$ in the body

$$\neg(\forall\alpha_1'\leqslant\mathscr{F}(\rho_{11}) .. \alpha_n'\leqslant\mathscr{F}(\rho_{1n}).\ \neg\mathscr{F}(\rho_{11}))$$

of the left-hand side.

Swapping right- and left-hand sides again yields a statement of the same form as the original, where the appropriate instances of $\mathscr{F}(\rho_{11}) .. \mathscr{F}(\rho_{1n})$ appear as the bounds of the outer quantifiers on the right,

$$\begin{aligned}\vdash\ldots\leqslant(\forall\alpha_1'\leqslant\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\,\mathscr{F}(\rho_{11}) .. \\ \alpha_n'\leqslant\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\,\mathscr{F}(\rho_{1n}). \\ \neg\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\,\mathscr{F}(\rho_{11}),\end{aligned}$$

i.e.,

$$\begin{aligned}\vdash \ldots \leqslant (\forall \gamma_1 \leqslant \{\mathscr{F}(\rho_1)/\alpha_1 \,..\, \mathscr{F}(\rho_n)/\alpha_n\}\ \mathscr{F}(\rho_{11})\,..\\ \gamma_n \leqslant \{\mathscr{F}(\rho_1)/\alpha_1 \,..\, \mathscr{F}(\rho_n)/\alpha_n\}\ \mathscr{F}(\rho_{1n}).\\ \neg\, \{\mathscr{F}(\rho_1)/\alpha_1 \,..\, \mathscr{F}(\rho_n)/\alpha_n\}\ \mathscr{F}(\rho_{11}).\end{aligned}$$

To get back to a statement of exactly the same form as the original, one further piece of mechanism is required: besides the n variables used to store the old state of the registers, a variable γ_0 holds the original value of the entire left-hand side of $\mathscr{F}(R)$. This variable is used at the end of a cycle to set up the left hand side of the statement encoding the next state of the rowing machine.

8.1. DEFINITION. Let ρ be a row of width n. The $F^{\mathrm{F}}_{\leqslant}$*-translation* of ρ, written $\mathscr{F}(\rho)$, is the n-negative type

$$\mathscr{F}(\rho) = \begin{cases} \alpha_i & \text{if} \quad \rho = \alpha_i \\ \forall \gamma_0, \alpha_1 \,..\, \alpha_n.\ \neg\, (\forall \gamma_0' \leqslant \gamma_0,\ \alpha_1' \leqslant \mathscr{F}(\rho_1) \,..\, \alpha_n' \leqslant \mathscr{F}(\rho_n).\ \neg\, \mathscr{F}(\rho_1)) & \\ & \text{if} \quad \rho = [\alpha_1 \,..\, \alpha_n]\langle \rho_1 \,..\, \rho_n \rangle \\ \forall \gamma_0, \alpha_1 \,..\, \alpha_n.\ \neg\, \mathrm{Top} & \text{if} \quad \rho = \mathrm{HALT}, \end{cases}$$

where γ_0, γ_0', and $\alpha_1' \,..\, \alpha_n'$ are fresh variables.

8.2. *Fact.* 1. The free variables of ρ coincide with the free type variables of $\mathscr{F}(\rho)$.

2. $\mathscr{F}(\{\rho_1/\alpha_1 \,..\, \rho_n/\alpha_n\}\,\rho) = \{\mathscr{F}(\rho_1)/\alpha_1 \,..\, \mathscr{F}(\rho_n)/\alpha_n\}\ \mathscr{F}(\rho)$.

8.3. DEFINITION. Let $R = \langle \rho_1 \,..\, \rho_n \rangle$ be a rowing machine. The $F^{\mathrm{F}}_{\leqslant}$*-translation* of R, written $\mathscr{F}(R)$, is the $F^{\mathrm{F}}_{\leqslant}$ statement

$$\vdash\ \sigma\ \leqslant\ \forall \gamma_0 \leqslant \sigma,\ \gamma_1 \leqslant \mathscr{F}(\rho_1) \,..\, \gamma_n \leqslant \mathscr{F}(\rho_n).\ \neg\, \mathscr{F}(\rho_1),$$

where

$$\sigma \equiv \forall \gamma_0, \gamma_1 \,..\, \gamma_n.\ \neg\, (\forall \gamma_0' \leqslant \gamma_0,\ \gamma_1' \leqslant \gamma_1 \,..\, \gamma_n' \leqslant \gamma_n.\ \neg\, \gamma_0)$$

and $\gamma_0, \gamma_1, \ldots, \gamma_n$ are fresh type variables. (Note that σ occurs on both sides of $\mathscr{F}(R)$.)

8.4. *Fact.* This definition is proper—i.e., $\mathscr{F}(R)$ is a closed $F^{\mathrm{F}}_{\leqslant}$-statement for every rowing machine R.

8.5. LEMMA. *If* $R \longrightarrow_{\mathrm{R}} R'$, *then* $\mathscr{F}(R) \xrightarrow{+}_{\mathrm{F}} \mathscr{F}(R')$.

Proof. By the definition of the elaboration function for rowing machines,

$$R \equiv \langle \rho_1 .. \rho_n \rangle,$$

where

$$\rho_1 \equiv [\alpha_1 .. \alpha_n] \langle \rho_{11} .. \rho_{1n} \rangle,$$

and

$$R' \equiv \langle \{\rho_1/\alpha_1 .. \rho_n/\alpha_n\} \rho_{11} .. \{\rho_1/\alpha_1 .. \rho_n/\alpha_n\} \rho_{1n} \rangle.$$

Let

$$\sigma \equiv \forall \gamma_0, \gamma_1 .. \gamma_n. \neg (\forall \gamma_0' \leqslant \gamma_0, \gamma_1' \leqslant \gamma_1 .. \gamma_n' \leqslant \gamma_n. \neg \gamma_0).$$

Now calculate as follows:

$$\begin{aligned}
& \mathscr{F}(R) \\
\equiv\ & \vdash \sigma \\
\leqslant\ & \forall \gamma_0 \leqslant \sigma, \gamma_1 \leqslant \mathscr{F}(\rho_1) .. \gamma_n \leqslant \mathscr{F}(\rho_n). \neg \mathscr{F}(\rho_1) \\
\equiv\ & \vdash \forall \gamma_0, \gamma_1 .. \gamma_n. \neg (\forall \gamma_0' \leqslant \gamma_0, \gamma_1' \leqslant \gamma_1 .. \gamma_n' \leqslant \gamma_n. \neg \gamma_0) \\
\leqslant\ & \forall \gamma_0 \leqslant \sigma, \gamma_1 \leqslant \mathscr{F}(\rho_1) .. \gamma_n \leqslant \mathscr{F}(\rho_n). \neg \mathscr{F}(\rho_1) \\
\longrightarrow_{\mathrm{F}}\ & \vdash \{\sigma/\gamma_0, \mathscr{F}(\rho_1)/\gamma_1 .. \mathscr{F}(\rho_n)/\gamma_n\} \mathscr{F}(\rho_1) \\
\leqslant\ & \{\sigma/\gamma_0, \mathscr{F}(\rho_1)/\gamma_1 .. \mathscr{F}(\rho_n)/\gamma_n\}(\forall \gamma_0' \leqslant \gamma_0, \gamma_1' \leqslant \gamma_1 .. \gamma_n' \leqslant \gamma_n. \neg \gamma_0) \\
\equiv\ & \vdash \mathscr{F}(\rho_1) \\
\leqslant\ & \forall \gamma_0' \leqslant \sigma, \gamma_1' \leqslant \mathscr{F}(\rho_1) .. \gamma_n' \leqslant \mathscr{F}(\rho_n). \neg \sigma \\
\equiv\ & \vdash \forall \gamma_0, \alpha_1 .. \alpha_n. \neg (\forall \gamma_0' \leqslant \gamma_0, \alpha_1' \leqslant \mathscr{F}(\rho_{11}) .. \alpha_n' \leqslant \mathscr{F}(\rho_{1n}). \neg \mathscr{F}(\rho_{11})) \\
\leqslant\ & \forall \gamma_0' \leqslant \sigma, \gamma_1' \leqslant \mathscr{F}(\rho_1) .. \gamma_n' \leqslant \mathscr{F}(\rho_n). \neg \sigma \\
\equiv\ & \vdash \forall \gamma_0, \alpha_1 .. \alpha_n. \neg (\forall \gamma_0' \leqslant \gamma_0, \alpha_1' \leqslant \mathscr{F}(\rho_{11}) .. \alpha_n' \leqslant \mathscr{F}(\rho_{1n}). \neg \mathscr{F}(\rho_{11})) \\
\leqslant\ & \forall \gamma_0 \leqslant \sigma, \alpha_1 \leqslant \mathscr{F}(\rho_1) .. \alpha_n \leqslant \mathscr{F}(\rho_n). \neg \sigma \\
\longrightarrow_{\mathrm{F}}\ & \vdash \{\sigma/\gamma_0, \mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\} \sigma \\
\leqslant\ & \{\sigma/\gamma_0, \mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\} \\
& \qquad (\forall \gamma_0' \leqslant \gamma_0, \alpha_1' \leqslant \mathscr{F}(\rho_{11}) .. \alpha_n' \leqslant \mathscr{F}(\rho_{1n}). \neg \mathscr{F}(\rho_{11}))
\end{aligned}$$

$$
\begin{aligned}
&\equiv \vdash \sigma \\
&\quad \leqslant \forall \gamma_0' \leqslant \sigma, \alpha_1' \leqslant (\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\, \mathscr{F}(\rho_{11})) .. \\
&\qquad\qquad \alpha_n' \leqslant (\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\, \mathscr{F}(\rho_{1n})). \\
&\qquad\qquad\quad \neg (\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\, \mathscr{F}(\rho_{11})) \\
&\equiv \vdash \sigma \\
&\quad \leqslant \forall \gamma_0 \leqslant \sigma, \gamma_1 \leqslant (\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\, \mathscr{F}(\rho_{11})) .. \\
&\qquad\qquad \gamma_n \leqslant (\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\, \mathscr{F}(\rho_{1n})). \\
&\qquad\qquad\quad \neg (\{\mathscr{F}(\rho_1)/\alpha_1 .. \mathscr{F}(\rho_n)/\alpha_n\}\, \mathscr{F}(\rho_{11})) \\
&\equiv \mathscr{F}(R'). \quad \blacksquare
\end{aligned}
$$

8.6. LEMMA. *If* $R \equiv \langle \text{HALT}, \rho_2 .. \rho_n \rangle$, *then* $\mathscr{F}(R)$ *is derivable in* $F^{F}_{\leqslant}$.

Proof. Let

$$\sigma \equiv \forall \gamma_0, \gamma_1 .. \gamma_n. \neg (\forall \gamma_0' \leqslant \gamma_0, \gamma_1' \leqslant \gamma_1 .. \gamma_n' \leqslant \gamma_n. \neg \gamma_0).$$

Then

$$
\begin{aligned}
&\mathscr{F}(R) \\
&\quad \equiv \vdash \sigma \\
&\qquad \leqslant \forall \gamma_0 \leqslant \sigma, \gamma_1 \leqslant \mathscr{F}(\text{HALT}) .. \gamma_n \leqslant \mathscr{F}(\rho_n). \neg \mathscr{F}(\text{HALT}) \\
&\quad \equiv \vdash \forall \gamma_0, \gamma_1 .. \gamma_n. \neg (\forall \gamma_0' \leqslant \gamma_0, \gamma_1' \leqslant \gamma_1 .. \gamma_n' \leqslant \gamma_n. \neg \gamma_0) \\
&\qquad \leqslant \forall \gamma_0 \leqslant \sigma, \gamma_1 \leqslant \mathscr{F}(\text{HALT}) .. \gamma_n \leqslant \mathscr{F}(\rho_n). \neg \mathscr{F}(\text{HALT}) \\
&\longrightarrow_{\text{F}} \vdash \{\sigma/\gamma_0, \mathscr{F}(\text{HALT})/\gamma_1 .. \mathscr{F}(\rho_n)/\gamma_n\}\, \mathscr{F}(\text{HALT}) \\
&\qquad \leqslant \{\sigma/\gamma_0, \mathscr{F}(\text{HALT})/\gamma_1 .. \mathscr{F}(\rho_n)/\gamma_n\}(\forall \gamma_0' \leqslant \gamma_0, \gamma_1' \leqslant \gamma_1 .. \gamma_n' \leqslant \gamma_n. \neg \gamma_0) \\
&\quad \equiv \vdash \mathscr{F}(\text{HALT}) \\
&\qquad \leqslant \forall \gamma_0' \leqslant \sigma, \gamma_1' \leqslant \mathscr{F}(\text{HALT}) .. \gamma_n' \leqslant \mathscr{F}(\rho_n). \neg \sigma \\
&\quad \equiv \vdash \forall \gamma_0, \alpha_1 .. \alpha_n. \neg \text{Top} \\
&\qquad \leqslant \forall \gamma_0' \leqslant \sigma, \gamma_1' \leqslant \mathscr{F}(\text{HALT}) .. \gamma_n' \leqslant \mathscr{F}(\rho_n). \neg \sigma \\
&\longrightarrow_{\text{F}} \vdash \sigma \\
&\qquad \leqslant \text{Top},
\end{aligned}
$$

which is an instance of FTOP. ∎

8.7. COROLLARY. *The rowing machine R halts iff $\mathscr{F}(R)$ is derivable in $F^{F}_{\leqslant}$.*

8.8. *Remark.* It is natural to ask whether Ghelli's nonterminating example (4.1) is the image of some rowing machine under this translation. The answer is "almost." Although the style of divergence in Ghelli's example is suggestive of the stepping behavior of translated rowing machines, every rowing machine translation involves a type σ of an appropriate form, which is not present in Ghelli's example. The pattern of regress is similar, but the cycles in Ghelli's example are shorter. Indeed Ghelli (1992) has shown that his example is, in a certain sense, the minimal input that causes nontermination.

9. TWO-COUNTER MACHINES

This section reviews the definition of two-counter Turing machines; see, e.g., Hopcroft and Ullman (1979) for more details.

9.1. DEFINITION. A *two-counter machine* is a tuple

$$\langle \mathrm{PC}, A, B, I_1..I_w \rangle,$$

where A and B are nonnegative numbers and PC and I_1 through I_w are instructions of the forms

$$\begin{aligned} &\text{INCA}\Rightarrow m \\ &\text{INCB}\Rightarrow m \\ &\text{TSTA}\Rightarrow m/n \\ &\text{TSTB}\Rightarrow m/n \\ &\text{HALT} \end{aligned}$$

with m and n in the range 1 to w.

9.2. DEFINITION. The *elaboration function* $\mathscr{E}$ for two-counter machines is the partial function mapping $T = \langle \mathrm{PC}, A, B, I_1..I_w \rangle$ to

$$\mathscr{E}(T) = \begin{cases} \langle I_m, A+1, B, I_1..I_w \rangle & \text{if } \mathrm{PC} \equiv \text{INCA}\Rightarrow m \\ \langle I_m, A, B+1, I_1..I_w \rangle & \text{if } \mathrm{PC} \equiv \text{INCB}\Rightarrow m \\ \langle I_m, A, B, I_1..I_w \rangle & \text{if } \mathrm{PC} \equiv \text{TSTA}\Rightarrow m/n \text{ and } A = 0 \\ \langle I_n, A-1, B, I_1..I_w \rangle & \text{if } \mathrm{PC} \equiv \text{TSTA}\Rightarrow m/n \text{ and } A > 0 \\ \langle I_m, A, B, I_1..I_w \rangle & \text{if } \mathrm{PC} \equiv \text{TSTB}\Rightarrow m/n \text{ and } B = 0 \\ \langle I_n, A, B-1, I_1..I_w \rangle & \text{if } \mathrm{PC} \equiv \text{TSTB}\Rightarrow m/n \text{ and } B > 0 \\ \text{undefined} & \text{if } \mathrm{PC} \equiv \text{HALT}. \end{cases}$$

9.3. *Convention.* For the following examples, it is convenient to assign alphabetic labels to the instructions of a program. By convention, the instruction with label START is used as the initial PC, and the initial value in both registers is 0.

9.4. EXAMPLE. This program loads 5 into register A and 3 into register B, then compares A and B for equality by repeatedly decrementing them until one or both become zero; if both do so on the same iteration, the program halts; otherwise it goes into an infinite loop.

```
START   INCA⇒I1
I1      INCA⇒I2
I2      INCA⇒I3
I3      INCA⇒I4
I4      INCA⇒J0
J0      INCB⇒J1
J1      INCB⇒J2
J2      INCB⇒LL
LL      TSTA⇒AZ/AS
AZ      TSTB⇒AZBZ/AZBS
AS      TSTB⇒ASBZ/LL
AZBZ    HALT
AZBS    INCA⇒AZBS
ASBZ    INCA⇒ASBZ.
```

9.5. DEFINITION. A two-counter machine T *halts* if $T \xrightarrow{*}_{\mathrm{T}} T'$ for some machine $T' \equiv \langle \mathrm{HALT}, A', B', I_1 .. I_w \rangle$.

9.6. FACT. *The halting problem for two-counter machines is undecidable.*

Proof Sketch. Hopcroft and Ullman (1979, pp. 171–173) show that a similar formulation of two-counter machines is Turing-equivalent. (Their two-counter machines have test instructions that do not change the contents of the register being tested, and separate decrement instructions. It is easy to check that this formulation and the one used here are inter-encodable.) ∎

10. Encoding Two-Counter Machines as Rowing Machines

We can now finish the proof of the undecidability of $F_{\leqslant}$ subtyping by showing that any two-counter machine T can be encoded as a rowing machine $\mathscr{R}(T)$ such that T halts iff $\mathscr{R}(T)$ does.

The main trick of the encoding lies in the representation of natural numbers as rows. Each number n is encoded as a *program* (i.e., a row) that, when executed, branches indirectly through one of two registers whose contents have been set beforehand to appropriate destinations for the zero and nonzero cases of a test; in other words, n itself encapsulates the behavior of the test instruction on a register containing n. The increment operation simply builds a new program of this sort from an existing one. The new program saves a pointer to the present contents of the register in a local variable so that it can restore the old value (i.e., one less than its own value) before executing the branch.

The encoding $\mathscr{R}(T)$ of a two-counter machine $T \equiv \langle \mathrm{PC}, A, B, I_1 .. I_w \rangle$ comprises the following registers:

#1	$\mathscr{R}^w(\mathrm{PC})$
#2	$\mathscr{R}^w_A(A)$
#3	$\mathscr{R}^w_B(B)$
#4	address register for zero branches
#5	address register for nonzero branches
#6	$\mathscr{R}^w(I_1)$
...	
$\#6 + w - 1$	$\mathscr{R}^w(I_w)$.

We use four translation functions for the various components:

1. $\mathscr{R}(T)$ is the encoding of a two-counter machine T as a rowing machine of width $w + 5$;

2. $\mathscr{R}^w(I)$ is the encoding of a two-counter instruction I as a row of width $w + 5$;

3. $\mathscr{R}^w_A(n)$ is the encoding of the natural number n, when it appears as the contents of register A, as a row of width $w + 5$;

4. $\mathscr{R}^w_B(n)$ is the encoding of the natural number n, when it appears as the contents of register B, as a row of width $w + 5$.

10.1. DEFINITION. The *row-encoding* (for w instructions) of a natural number n in register A, written $\mathscr{R}^w_A(n)$, is defined as follows:

$$\mathscr{R}^w_A(0) = \langle \#4, \cdots, —, \text{HALT}, \text{HALT}, \underbrace{— \,..\, —}_{w \text{ times}} \rangle$$

$$\mathscr{R}^w_A(n+1) = \langle \#5, \mathscr{R}^w_A(n), —, \text{HALT}, \text{HALT}, \underbrace{— \,..\, —}_{w \text{ times}} \rangle.$$

The row-encoding (for w instructions) of a natural number n in register B, written $\mathscr{R}^w_B(n)$, is defined as follows:

$$\mathscr{R}^w_B(0) = \langle \#4, —, —, \text{HALT}, \text{HALT}, \underbrace{— \,..\, —}_{w \text{ times}} \rangle$$

$$\mathscr{R}^w_B(n+1) = \langle \#5, —, \mathscr{R}^w_B(n), \text{HALT}, \text{HALT}, \underbrace{— \,..\, —}_{w \text{ times}} \rangle.$$

10.2. DEFINITION. The *row-encoding* (for w instructions) of an instruction I, written $\mathscr{R}^w(I)$, is defined as follows:

$$\mathscr{R}^w(\text{INCA}\Rightarrow m) = \langle \#m+5, \langle \#5, \#\#2, —, \text{HALT}, \text{HALT}, \cdots .. \cdots \rangle, —,$$
$$\text{HALT}, \text{HALT}, — \,..\, — \rangle$$

$$\mathscr{R}^w(\text{INCB}\Rightarrow m) = \langle \#m+5, \cdots, \langle \#5, \cdots, \#\#3, \text{HALT}, \text{HALT}, — \,..\, — \rangle,$$
$$\text{HALT}, \text{HALT}, — \,..\, — \rangle$$

$$\mathscr{R}^w(\text{TSTA}\Rightarrow m/n) = \langle \#2, \cdots, —, \#m+5, \#n+5, — \,..\, — \rangle$$

$$\mathscr{R}^w(\text{TSTB}\Rightarrow m/n) = \langle \#3, \cdots, \cdots, \#m+5, \#n+5, — \,..\, — \rangle$$

$$\mathscr{R}^w(\text{HALT}) = \langle \text{HALT}, —, —, \text{HALT}, \text{HALT}, — \,..\, — \rangle.$$

10.3. DEFINITION. Let $T = \langle \text{PC}, A, B, I_1..I_w \rangle$ be a two-counter machine. The *row-encoding* of T, written $\mathscr{R}(T)$, is the rowing machine of width $w+5$ defined as follows:

$$\mathscr{R}(T) = \langle \mathscr{R}^w(\text{PC}), \mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \text{HALT}, \text{HALT}, \mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle.$$

10.4. LEMMA. *If* $T \longrightarrow_T T'$, *then* $\mathscr{R}(T) \xrightarrow{+}_R \mathscr{R}(T')$.

Proof. Let $T = \langle \text{PC}, A, B, I_1..I_w \rangle$. Proceed by cases on the form of PC.

Case:

$$\text{PC} = \text{INCA}\Rightarrow m.$$

Then $T' = \langle I_m, A+1, B, I_1 .. I_w \rangle$. Calculate as follows:

$$
\begin{aligned}
& \mathscr{R}(T) \\
\equiv\ & \langle \langle \#m+5, \langle \#5, \#\#2, —, \text{HALT}, \text{HALT}, — .. — \rangle, \\
& \quad \cdots, \text{HALT}, \text{HALT}, — .. — \rangle, \\
& \mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \\
& \text{HALT}, \text{HALT} \\
& \mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\longrightarrow_R\ & \langle \mathscr{R}^w(I_m), \\
& \langle \#5, \mathscr{R}^w_A(A), —, \text{HALT}, \text{HALT}, — .. — \rangle, \mathscr{R}^w_B(B), \\
& \text{HALT}, \text{HALT}, \\
& \mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\equiv\ & \mathscr{R}(T').
\end{aligned}
$$

Case:

$$\text{PC} = \text{INCB} \Rightarrow m.$$

Similar.

Case:

$$\text{PC} = \text{TSTA} \Rightarrow m/n.$$

Calculate as follows:

$$
\begin{aligned}
& \mathscr{R}(T) \\
\equiv\ & \langle \langle \#2, —, —, \#m+5, \#n+5, — .. — \rangle, \\
& \mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \\
& \text{HALT}, \text{HALT} \\
& \mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\longrightarrow_R\ & \langle \mathscr{R}^w_A(A), \\
& \mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \\
& \mathscr{R}^w(I_m), \mathscr{R}^w(I_n), \\
& \mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle.
\end{aligned}
$$

There are two subcases to consider:

Subcase:

$$A = 0.$$

Then

$$\mathscr{R}^w_A(A) = \langle \#4, —, —, \text{HALT}, \text{HALT}, — .. — \rangle$$
$$T' = \langle I_m, A, B, I_1 .. I_w \rangle.$$

Continue calculating as follows:

$$\begin{aligned}
&\langle \langle \#4, —, —, \text{HALT}, \text{HALT}, — .. — \rangle, \\
&\mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \\
&\mathscr{R}^w(I_m), \mathscr{R}^w(I_n), \\
&\mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\longrightarrow_R\ &\langle \mathscr{R}^w(I_m), \\
&\mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \\
&\text{HALT}, \text{HALT}, \\
&\mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\equiv\ &\mathscr{R}(T').
\end{aligned}$$

Subcase:

$$A > 0.$$

Then

$$\mathscr{R}^w_A(A) = \langle \#5, \mathscr{R}^w_A(A-1), —, \text{HALT}, \text{HALT}, — .. — \rangle$$
$$T' = \langle I_n, A-1, B, I_1 .. I_w \rangle.$$

Continue calculating as follows:

$$\begin{aligned}
&\langle \langle \#5, \mathscr{R}^w_A(A-1), —, \text{HALT}, \text{HALT}, — .. — \rangle, \\
&\mathscr{R}^w_A(A), \mathscr{R}^w_B(B), \\
&\mathscr{R}^w(I_m), \mathscr{R}^w(I_n), \\
&\mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\longrightarrow_R\ &\langle \mathscr{R}^w(I_n), \\
&\mathscr{R}^w_A(A-1), \mathscr{R}^w_B(B), \\
&\text{HALT}, \text{HALT}, \\
&\mathscr{R}^w(I_1) .. \mathscr{R}^w(I_w) \rangle \\
\equiv\ &\mathscr{R}(T').
\end{aligned}$$

Case:

$$\text{PC} = \text{TSTB} \Rightarrow m/n.$$

Similar.

Case:

$$\text{PC} = \text{HALT}.$$

Cannot happen. ∎

10.5. LEMMA. *If* $T = \langle \text{HALT}, A, B, I_1 .. I_w \rangle$, *then* $\mathscr{R}(T)$ *halts.*

Proof. Immediate. ∎

10.6. COROLLARY. *T halts iff* $\mathscr{R}(T)$ *does*

11. UNDECIDABILITY

11.1. THEOREM. *The* $F_{\leqslant}$ *subtyping relation is undecidable.*

Proof. Assume, for a contradiction, that we had a total-recursive procedure for testing the derivability of subtyping statements in $F_{\leqslant}$. Then to decide whether a two-counter machine T halts, we could use this procedure to test whether $\mathscr{F}(\mathscr{R}(T))$ is derivable, since

T halts	
iff $\mathscr{R}(T)$ halts	by Corollary 10.6
iff $\mathscr{F}(\mathscr{R}(T))$ is derivable in $F^{\text{F}}_{\leqslant}$	by Corollary 8.7
iff $\mathscr{F}(\mathscr{R}(T))$ is derivable in $F^{\text{D}}_{\leqslant}$	by Lemma 6.10
iff $\mathscr{F}(\mathscr{R}(T))$ is derivable in $F^{\text{N}}_{\leqslant}$	by Lemma 5.9
iff $\mathscr{F}(\mathscr{R}(T))$ is derivable in $F_{\leqslant}$	by Lemma 3.2. ∎

From the undecidability of $F_{\leqslant}$ subtyping, the undecidability of typechecking follows immediately: we need only show how to write down a term that is well typed iff a given subtyping statement

$$\vdash \sigma \leqslant \tau$$

is derivable. One such term is

$$\lambda f{:}\tau \to \text{Top}.\ \lambda a{:}\sigma.\ f\ a.$$

12. Discussion

The undecidability of $F_{\leqslant}$ came as a surprise to many who have studied, extended, and applied it since its introduction in 1985. But it seems probable that language designs and implementations based on $F_{\leqslant}$ will not be greatly affected by this discovery. Here are some reasons for optimism:

1. The algorithm has been used for several years now without any sign of misbehavior in any situation arising in practice. Indeed, constructing even the simplest nonterminating example (Ghelli, 1992) requires a contortion that is difficult to imagine anyone performing by accident.

2. A number of useful fragments of $F_{\leqslant}$ are easily shown to be decidable. For example:

- The prenex fragment, where all quantifiers appear at the outside and quantifiers are instantiated only at monotypes (types containing no quantifiers).

- A predicative fragment where types are stratified into universes and the bound of a quantified type lives in a lower universe than the quantified type itself.

- A fragment where bounds of quantifiers are not allowed to contain Top (Katiyar and Sankar, 1992).

- Cardelli and Wegner's original formulation where the bounds of two quantified types must be identical in order for one to be a subtype of the other.

Though semantically unappealing, this formulation of $F_{\leqslant}$ appears strong enough to include essentially all useful programming examples. The only known examples that require the more general quantifier subtyping rule and those involving bounded existential types, which correspond to "partially abstract types" (cf. (Cardelli and Wegner, 1985)) under Mitchell and Plotkin's correspondence between abstract types and existential types (1988). Partially abstract types are a generalization of abstract types where some of the structure of the representation type is known but its exact identity remains hidden. Interesting subtype relations between partially abstract types can only arise from the full $F_{\leqslant}$ quantifier subtyping rule.

3. The algorithms for these fragments are essentially identical to the algorithm $F^{\mathrm{N}}_{\leqslant}$.

4. On well-typed expressions, a type synthesis algorithm based on $F^{\mathrm{N}}_{\leqslant}$ is guaranteed to terminate, since it will only ask subtyping questions to which the answer is "yes."

ACKNOWLEDGMENTS

I am grateful for discussions with John Reynolds, Robert Harper, Luca Cardelli, Giorgio Ghelli, Daniel Sleator, and Tim Freeman. Numerous comments from two referees have greatly improved the presentation. This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order 7597; in part by the Office of Naval Research under Contract N00013-84-K-0415; in part by the National Science Foundation under Contract CR-8922109; and in part by Siemens.

RECEIVED March 3, 1992; FINAL MANUSCRIPT RECEIVED March 18, 1993

REFERENCES

BREAZU-TANNEN, V., COQUAND, T., GUNTER, C., AND SCEDROV, A. (1991), Inheritance as implicit coercion, *Inform. and Comput.* **93**, 172–221.

BRUCE, K. B., AND LONGO, G. (1990), A modest model of records, inheritance, and bounded quantification, *Inform. and Comput.* **87**, 196–240. To appear in (Gunter and Mitchell, 1994). An earlier version appeared in the Proceedings of the IEEE Symposium on Logic in Computer Science, 1988.

BRUCE, K., AND MITCHELL, J. (1992), PER models of subtyping, recursive types and higher-order polymorphism, *in* "Proceedings of the Nineteenth ACM Symposium on Principles of Programming Languages, Albequerque, NM."

BRUCE, K. B. (1991), The equivalence of two semantic definitions for inheritance in object-oriented languages, *in* "Proceedings of Mathematical Foundations of Programming Semantics, Pittsburgh, PA."

BRUCE, K. B. (1992), "A Paradigmatic Object-Oriented Language: Design, Static Typing and Semantics," Technical Report CS-92-01, Williams College.

BRUCE, K. B. (1993), Safe type checking in a statically typed object-oriented programming language, *in* "Proceedings of the Twentieth ACM Symposium on Principles of Programming Languages."

CANNING, P., COOK, W., WILL, W., AND OLTHOFF, W. (1989a), Interfaces for strongly-typed object-oriented programming, *in* "Object Oriented Programming: Systems, Languages, and Applications (Conference Proceedings)," pp. 457–467.

CANNING, P., COOK, W., WILL, W., AND OLTHOFF, W. (1989b), *F*-bounded quantification for object-oriented programming, *in* "Fourth International Conference on Functional Programming Languages and Computer Architecture," pp. 273–280.

CARDELLI, L., AND LONGO, G. (1991), A semantic basis for Quest, *J. Functional Programming* **1**(4), 417–458. Preliminary version in "ACM Conference on Lisp and Functional Programming," June 1990. Also available as DEC SRC Research Report 55, Feb. 1990.

CARDELLI, L., AND MITCHELL, J. (1991), Operations on records, *Math. Structures Comput. Sci.* **1**, 3–48. To appear in (Gunter and Mitchell, 1994); also available as DEC Systems Research Center Research Report #48, August, 1989, and in the proceedings of MFPS '89, Springer LNCS volume 442.

CARDELLI, L., AND WEGNER, P. (1985), On understanding types, data abstraction, and polymorphism, *Comput. Surv.* **17**(4).

CARDELLI, L., MARTINI, S., MITCHELL, J. C., AND SCEDROV, A. (1991), An extension of system *F* with subtyping, *in* Ito and Meyer (1991, pp. 750–770).

CARDELLI, L. (1988a), A semantics of multiple inheritance, *Inform. and Comput.* **76**, 138–164. Preliminary version in "Semantics of Data Types" (Kahn, MacQueen, and Plotkin, Eds.), Lecture Notes in Computer Science, Vol. 173, Springer-Verlag, Berlin/New York.

CARDELLI, L. (1988b), Structural subtyping and the notion of power type, *in* "Proceedings of the 15th ACM Symposium on Principles of Programming Languages, San Diego," pp. 70–79.

CARDELLI, L. (1991a), *F*-sub, the system, unpublished manuscript.

CARDELLI, L. (1991b), Typeful programming, *in* "Formal Description of Programming Concepts" (E. J. Neuhold and M. Paul, Eds.), Springer-Verlag, Berlin/New York. An earlier version appeared as DEC Systems Research Center Research Report 45, 1989.

CARDELLI, L. (1992), "Extensible Records in a Pure Calculus of Subtyping," Research Report 81, DEC Systems Research Center. To appear in (Gunter and Mitchell, 1994).

CARDONE, F. (1989), Relational semantics for recursive types and bounded quantification, *in* "Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming, Stresa, Italy," pp. 164–178, Lecture Notes in Computer Science, Vol. 372, Springer-Verlag, Berlin/New York.

COOK, W. R., HILL, W. L., AND CANNING, P. S. (1990), Inheritance is not subtyping, *in* "Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco," pp. 125–135. To appear in (Gunter and Mitchell, 1994).

COPPO, M., DEZANI-CIANCAGLINI, M., AND VENNERI, B. (1980), Principal type schemes and lambda calculus semantics, *in* "To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism," pp. 535–560, Academic Press, New York.

CURIEN, P.-L., AND GHELLI, G. (1991), Subtyping + extensionality: Confluence of $\beta\eta$-reductions in $F_{\leqslant}$, *in* Ito and Meyer (1991, pp. 731–749).

CURIEN, P.-L., AND GHELLI, G. (1992), Coherence of subsumption: Minimum typing and type-checking in $F_{\leqslant}$, *Math. Structures Comput. Sci.* **2**, 55–91. To appear in (Gunter and Mitchell, 1994).

DE BRUIJN, N. G. (1972), Lambda-calculus notation with nameless dummies: A tool for automatic formula manipulation with application to the Church–Rosser theorem, *Indag. Math.* **34**(5), 381–392.

GHELLI, G., AND PIERCE, B. (1992), "Bounded Existentials and Minimal Typing," Draft technical report.

GHELLI, G. (1990), "Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism," Ph.D. Thesis, Università di Pisa; Technical Report TD-6/90, Dipartimento di Informatica, Università di Pisa.

GHELLI, G. (1991), Modelling features of object-oriented languages in second-order functional languages with subtypes, *in* "Foundations of Object-Oriented Programming" (J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds.), pp. 311–340, Lecture Notes in Computer Science, Vol. 489, Springer-Verlag, Berlin/New York.

GHELLI, G. (1992), Divergence of $F_{\leqslant}$ type checking, *Theoret. Comput. Sci.*, to appear.

GHELLI, G. (1993), Recursive types are not conservative over $F_{\leqslant}$, *in* "Typed Lambda Calculus and Applications."

GIRARD, J. Y. (1972), "Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur," Ph.D. Thesis, Université de Paris VII.

GUNTER, C. A., AND MITCHELL, J. C., Eds. (1994), "Theoretical Aspects de Object-Oriented Programming: Types, Semantics, and Language Design," MIT Press, Cambridge, MA.

HOFMANN, M., AND PIERCE, B. (1992), "An Abstract View of Objects and Subtyping (Preliminary Report)," Technical Report ECS-LFCS-92-226, University of Edinburgh.

HOPCROFT, J. E., AND ULLMAN, J. D. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA.

ITO, T., AND MEYER, A. R., Eds. (1991), "Theoretical Aspects of Computer Software, Sendai, Japan," Lecture Notes in Computer Science, No. 526, Springer-Verlag, Berlin/New York.

KATIYAR, D., AND SANKAR, S. (1992), Completely bounded quantification is decidable, *in* "Proceedings of the ACM SIGPLAN Workshop on ML and its Applications."

MARTINI, S. (1988), Bounded quantifiers have interval models, *in* "Proceedings of the ACM Conference on Lisp and Functional Programming, Snowbird, UT," pp. 174–183, Assoc. Comput. Mach.

MITCHELL, J., AND PLOTKIN, G. (1988), Abstract types have existential type, *ACM Trans. Programming Languages Systems* **10**(3).

MITCHELL, J., MELDAL, S., AND MADHAV, N. (1991), An extension of Standard ML modules with subtyping and inheritance, *in* "Proceedings of the Eighteenth ACM Symposium on Principles of Programming Languages, Orlando, FL," pp. 270–278.

MITCHELL, J. C. (1988), Polymorphic type inference and containment, *Inform. and Comput.* **76**, 211–249

PIERCE, B. C., AND TURNER, D. N. (1994), Simple type-theoretic foundations for object-oriented programming, *J. Functional Programming*, to appear. A preliminary version appeared in "Principles of Programming Languages" (1993) and as University of Edinburgh Technical Report ECS-LFCS-92-225, under the title "Object-Oriented Programming Without Recursive Types."

PIERCE, B. C. (1991), "Programming with Intersection Types and Bounded Polymorphism," Ph.D. Thesis, Carnegie–Mellon University. Available as School of Computer Science Technical Report CMU-CS-91-205.

PIERCE, B. C. (1993), Intersection types and bounded polymorphism, *in* "Conference on Typed Lambda Calculi and Applications," to appear. Preliminary version available as University of Edinburgh Technical Report ECS-LFCS-92-200.

REYNOLDS, J. (1974), Towards a theory of type structure, *in* "Proc. Colloque sur la Programmation, New York," pp. 408–425, Lecture Notes in Computer Science, Vol. 19, Springer-Verlag, Berlin/New York.

REYNOLDS, J. C. (1988), "Preliminary Design of the Programming Language Forsythe," Technical Report CMU-CS-88-159, Carnegie–Mellon University.

Printed in Belgium
Uitgever: Academic Press, Inc.
Verantwoordelijke uitgever voor België:
Hubert Van Maele
Altenastraat 20, B-8310 Sint-Kruis